TEC | Tecnológico de Costa Rica

Factorizando numeros

Taller de programación Grupo 21
Escuela de computación
Ingeniería en Computación
Profesor Edgar Rojas Muñoz
Marco Rodriguez Vargas

19 de abril de 2022

1 Resumen ejecutivo

La factorización de numeros es relevante porque sigue siendo un misterio en la actualidad. Aunque se han encontrado métodos para aproximar este proceso, aún no hay ninguna forma de hacerlo a grandes números. Muchas personas han dedicado su vida al estudio de los números primos por lo que es un tema que se ha explorado profundamente.

En el proyecto se plantea la solución de cinco formas en las que se puede llegar a dar una mejor aproximación. Se quiere factorizar números en sus factores primos. Esto es proveniente del Teorema del fundamental de la aritmética, el cúal dice que todo número natural tiene una factorización prina única.

La soluciones que se nos plantean son ir buscando factores uno a uno hasta la mitad del número en cuestión (de forma recursiva e iterativa), dividir por dos tantas veces como sea posible mientras el número en cuestión sea par y luego probar con los números impares hasta la raíz cuadrada del número en cuestión (de forma recursiva e iterativa) y antes de factorizar, generar la Criba de Eratóstenes y una vez esta sea generada, factorizar todos los números con la misma criba, utilizando únicamente los números primos (este puede hacerse de manera recursiva o iterativa)

La solución se estará evaluando mediante dos comparaciones, la primera será en la que todos realicen los mismos casos de prueba y la segunda será en la que cada uno llevará casos de prueba en la que se les lleve al límite. Despúes se compararán ambas gráficas para así ver cuál es mejor y si ambas tienen algún modo en común para de esta forma hallar el más eficiente. Los resultados indican que el modo más eficiente fue el Modo 4. Los demás demandaron mas tiempo en los casos de prueba que se les realizó. Lo que podría explicar su eficiencia es porque es iterativo y además con la división de números pares y con los impares hasta la raíz.

Los resultados fueron diferentes a los que se tenían pensados, se creía que la criba iba a ser más eficiente que los demás modos pero esta tiene un defecto, el cuál es el límite que se le asigne, ya que antes de empezar a factorizar los números esta generara la criba. Cuando este modo genera la criba pierde mucho tiempo y este tiempo aumentará con relación al limite que se le ponga a la función.

De acuerdo con todo lo anterior se puede concluir que aunque la factorización de números ha avanzado bastante en los ultimos años, su solución sigue siendo incierta. Probablemente se sigan encontrando posibles soluciones a este problema infinito, lo único cierto es que la matemática es un tema que no tiene final y sus preguntas siguen asombrando al ser humano.

2 Introducción

Uno de los mayores enigmas para el ser humano desde el comienzo de los tiempos ha sido la factorización de números. Cabe destacar que este acertijo se puede visualizar como números escondidos en una infinitud de números enteros, son los números primos. Este tipo de teoremas ha llegado a promover el desarrollo de la matemática desde el principio de los tiempos. En la actualidad no se ha encontrado algún algoritmo que llegue a resolver este problema de manera eficiente.

En este proyecto se plantea la solución de este problema, la factorización de números. Se intentará resolver para así intentar descubrir la forma de factorizar números y así descomponerlos en sus factores primos. De esta forma pretende encontrar una posible solución a esta interrogante. Se presentan 5 modos para así intentar resolver el dilema. Con el Modo 1 y el Modo 3 se resuelve de forma recursiva, el Modo 2 y el Modo 4 se resuelve de forma iterativa y el Modo 5 se plantea una solución con la Criba de Eratóstenes. Estos cinco modos entregan formas en las que se podría llegar a una solución eficiente.

Contents

3 Marco teórico

El lenguaje utilizado fue Python 3. Python es un lenguaje de programación de alto nivel y su propósito es bastante amplio. En un principio fue diseñado por Guido van Rossum en el año mil novecientos noventa y uno y desarrollado por Python Software Foundation. Fue desarrollado principalmente para enfatizar la facilidad para leer el código y su forma de construcción permite a los que la usen expresar funciones en menos líneas de código.

3.1 Bibliotecas

Sys: se utilizó para establecer un limite para las recursiones del código. Math: se usó para elevar al cuadrado un valor en el código.

3.2 IDE

El IDE utilizado en este caso fue Pycharm el cual es un interpretador de Python y se utilizó para Python 3. Un IDE es un ambiente de desarrollo para que Python se vea visualmente más accesible aunque es igual de funcional que el Python ordinario, solo existe para visualizar mejor los códigos y tener más facilidad para acceder a ellos

4 Descripción de la solución

4.1 Modo 1

En este caso se buscan factores 1 a 1 hasta la mitad del número que se esté analizando, esto de forma recursiva. Esto implica ir analizando cada factor hasta llegar al caso base. Su funcionamiento resultó sencillo, esto porque la función recursiva se brindó en el proyecto.

4.1.1 Problemas

Identificar que “fac” tenía que empezar por el numero 2 fue una de las principales dificultades, al igual que organizar los “prints” para que de esta forma se pudiera conseguir el formato de salida esperado para el proyecto. La solución para el primer problema fue razonar que el menor numero primo que se iba a necesitar era el dos (importante destacar que este es el único numero primo que es par)

4.2 Modo 2

En este caso se buscan factores 1 a 1 hasta la mitad del número que se esté analizando, esto de forma iterativa. Esto implica repetir varias veces este proceso hasta que se factorice. En este caso se hizo uso de ”While” para la generación del ciclo.

4.2.1 Problemas

No se presentaron problemas, solo otra vez recordar que “fac” tiene que ser 2 por lo explicado anteriormente. Como su estructura es similar al anterior código, colocar los “prints” resultó más intuitivo.

4.3 Modo 3

En este caso se divide el numero por 2 tantas veces como sea posible (siempre y cuando el número en cuestión fuera par) y después intentar con los impares hasta la raíz cuadrada del número en cuestión. De forma recursiva. Esto implica ir analizando cada factor hasta llegar al caso base. Su funcionamiento resultó sencillo, esto porque la función recursiva se brindó en el proyecto.

4.3.1 Problemas

Problemas: Como anteriormente se utilizó el factor 2 para las funciones se pensó que así iba a ser para el resto de los modos. La respuesta era que el factor pasaba a ser 3 por ser el siguiente número primo ya que este modo abarca la división de números pares

4.4 Modo 4

En este caso se divide el numero por 2 tantas veces como sea posible (siempre y cuando el número en cuestión fuera par) y después intentar con los impares hasta la raíz cuadrada del número en cuestión. De forma iterativa. Esto implica repetir varias veces este proceso hasta que se factorice. Esto implica repetir varias veces este proceso hasta que se factorice. En este caso se hizo uso de ”While” para la generación del ciclo.

4.4.1 Problemas

No se presentó ningún problema mas que darle importancia a “fac” es igual a 3 (variable).

4.5 Modo 5

Este consistía en antes de empezar la factorización de números, primero generar la Criba de Eratóstenes y una vez generada, factorizar todos los números con esa criba, utilizando solo números primos. (La Criba de Eratóstenes genera únicamente números primos)

4.5.1 Problemas

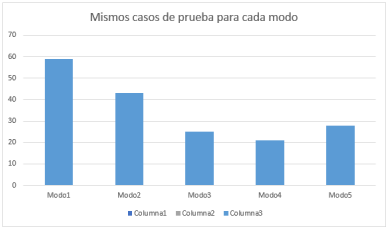
Este modo presento bastantes dificultades, aunque anteriormente se había programado para la tarea 1 (esto fue para practicar conceptos pero resultó beneficioso para el proyecto), eso hizo que esta segunda programación fuera menos compleja.

5 Resultados de pruebas

Para los casos de prueba se realizaron dos comparaciones diferentes (esto para tener dos puntos de comparación y de esa forma intentar aproximar una mejor conclusión).Esto siempre siendo pensado para que la comparación entre modos sea justa y realmente beneficiosa para todos y que no perjudique a ninguno por sus características de código.
Procesador: Intel i5 12th gen

6 Mismos casos de prueba

La primera comparación que se realizó fue los mismos casos de prueba para cada modo respectivamente. Los primeros dos fueron los mas tardados en esta prueba. Esto porque iban buscando factores 1 a 1, el Modo 1 tuvó una duración mayor porque la recursión no es tan eficiente en Python. Despúes se tiene que analizar que el Modo 3 y el Modo 4 lograron factorizar los numeros en la mitad del tiempo en comparación a los primeros dos modos pero aún la recursividad resultó mas lenta que la iteración. Por último la Criba de Eratóstenes fue la tercera más eficiente para los casos de prueba que se pusieron a disposición. De esta forma el Modo 4 fue el más eficiente en este caso, esto se debe a la iteración y a que se dividiera por 2 tantas veces como fuera posible.



Modo 1 con mismos casos de prueba			
Entrada	Casos de prueba	Salida	Segundos
10 1 100000	4	2 2	0,59s
	35	5 7	
	133	7 19	
	645	3 5 43	
	930	2 3 5 31	
	9356	2 2 2339	
	10230	2 3 5 11 31	
	20503	7 29 101	
	54300	2 2 3 5 5 181	
	57254	2 28627	

Table 1: Modo1

Modo 2 con mismos casos de prueba			
Entrada	Casos de prueba	Salida	Segundos
10 2 100000	4	2 2	0,43s
	35	5 7	
	133	7 19	
	645	3 5 43	
	930	2 3 5 31	
	9356	2 2 2339	
	10230	2 3 5 11 31	
	20503	7 29 101	
	54300	2 2 3 5 5 181	
	57254	2 28627	

Table 2: Modo2

Modo 3 con mismos casos de prueba			
Entrada	Casos de prueba	Salida	Segundos
10 2 100000	4	2 2	0,25s
	35	5 7	
	133	7 19	
	645	3 5 43	
	930	2 3 5 31	
	9356	2 2 2339	
	10230	2 3 5 11 31	
	20503	7 29 101	
	54300	2 2 3 5 5 181	
	57254	2 28627	

Table 3: Modo3

Modo 4 con mismos casos de prueba			
Entrada	Casos de prueba	Salida	Segundos
10 4 100000	4	2 2	0,21s
	35	5 7	
	133	7 19	
	645	3 5 43	
	930	2 3 5 31	
	9356	2 2 2339	
	10230	2 3 5 11 31	
	20503	7 29 101	
	54300	2 2 3 5 5 181	
	57254	2 28627	

Table 4: Modo4

Modo 5 con mismos casos de prueba			
Entrada	Casos de prueba	Salida	Segundos
10 5 100000	4	2 2	0,28s
	35	5 7	
	133	7 19	
	645	3 5 43	
	930	2 3 5 31	
	9356	2 2 2339	
	10230	2 3 5 11 31	
	20503	7 29 101	
	54300	2 2 3 5 5 181	
	57254	2 28627	

Table 5: Modo5

7 Diferentes casos de prueba

Esta comparación entregó datos similares al experimento de los mismos casos de prueba con pequeñas variaciones. Lo primero que se puede notar en la gráfica de casos en los que se establecía diferentes límites para cada modo fue que el Modo 5 pasó a ser el más lento esto por culpa del límite en su código (el cual generaba la criba, esto hace que se consuma bastante tiempo dependiendo de que tan grande sea el límite.) Lo segundo que se puede analizar es que la recursión en estos casos arrojó mejores tiempos que los modos que utilizaron iteración. Aún asi la mayor diferencia surge entre los primeros dos modos y los modos tres y cuatro los cuales gracias a dividir el número por dos tantas veces como fuera posible lograron disminuir el tiempo casi por la mitad, incluso llegando a números más grandes.

Modo 1 con diferentes casos de prueba			
Entrada	Casos de prueba	Salida	Segundos
10 1 1000000	4	2 2	0,59s
	35	5 7	
	133	7 19	
	645	3 5 43	
	930	2 3 5 31	
	9356	2 2 2339	
	10230	2 3 5 11 31	
	20503	7 29 101	
	54300	2 2 3 5 5 181	
	57254	2 28627	

Table 6: Modo1 intentando llegar al limite

Modo 2 con diferentes casos de prueba			
Entrada	Casos de prueba	Salida	Segundos
10 2 1000000000000000	4	2 2	0.62s
	3545	5 709	
	56443	56443	
	789564	2 2 3 19 3463	
	16677846	2 3 3 3 308849	
	268454464	2 2 2 2 2 2 4194601	
	3456355643	13 101 2632411	
	68796634563	3 3 139 641 85793	
	79030593560	2 2 2 5 12907 153077	
	90021059050	2 5 5 13 29 4775653	

Table 7: Modo2 intentando llegar al limite

Modo 3 con diferentes casos de prueba			
Entrada	Casos de prueba	Salida	Segundos
10 3 1000000000000000	4	2 2	0.31s
	134124	2 2 3 11177	
	4356566	2 2178283	
	8939045	5 233 7673	
	35635345	5 29 53 4637	
	17469233	157 111269	
	19367684	2 2 7 31 53 421	
	267876583	13 619 33289	
	378833862	2 3 11 5739907	
	6042006594	2 3 3 3 3 13 2868949	

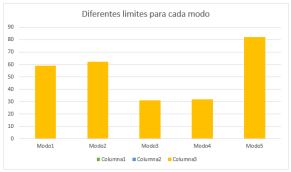
Table 8: Modo3 intentando llegar al limite

Modo 4 con diferentes casos de prueba			
Entrada	Casos de prueba	Salida	Segundos
10 4 1000000000000000	4	2 2	0.32s
	134124	2 2 3 11177	
	4356566	2 2178283	
	8939045	5 233 7673	
	35635345	5 29 53 4637	
	17469233	157 111269	
	19367684	2 2 7 31 53 421	
	267876583	13 619 33289	
	378833862	2 3 11 5739907	
	6042006594	2 3 3 3 3 13 2868949	

Table 9: Modo4 intentando llegar al limite

Modo 5 con diferentes casos de prueba			
Entrada	Casos de prueba	Salida	Segundos
10 5 100000000	4	2 2	0.82s
	134	2 67	
	435	3 5 29	
	8939	7 1277	
	35635	5 7127	
	17469	3 3 3 647	
	193674	2 3 13 13 191	
	267876	2 2 3 3 7 1063	
	378833	7 13 23 181	
	6042006	2 3 3 3 41 2729	

Table 10: Modo5 intentando llegar al limite



8 Conclusiones

Los resultados ayudaron a comprender mejor el tema de factorización de numeros primos, asímismo demostraron que su eficiencia depende del límite que se le asignara.En este caso se considera más rápido el Modo 4 ya que este fue el que tuvo mejores tiempos en ambos casos de prueba. Esto porque fue construido con iteración y además incorporando la división de números pares y luego probaba hasta la raíz cuadra del número. Además este proyecto ayudó a tener un mejor entendimiento de la factorización de números primos y las distintas formas que existen para aproximar los valores dentro de los limites establecidos. Por lo tanto se llega a la siguiente conclusión, el Modo 1 no es tan eficiente ya que su limite es bastante bajo esto porque Python tiene problemas para procesar la recursividad, el Modo 2 es un levemente más eficiente que el anterior por estar construido de forma iterativa, el Modo 3 es mejor que los anteriores aunque esté construido de forma recursiva, el Modo 4 es el que presentó mejores resultados con un buen limite y buenos tiempos por lo que se considera que es el más eficiente y por último el Modo 5 se pensaría que podría llegar a ser más eficiente pero en la forma que está construida pierde bastante tiempo a la hora de construir la criba. Por lo que se puede comentar que el Modo 4 fue el más útil.

9 Aprendizaje

Marco Rodríguez Vargas: en este proyecto logré practicar temas como iteración y while, además de otros temas como listas y funciones básicas en Python. Me ayudó a mejorar la forma en la que pienso el pseudocódigo e hizó que se volviera más fácil la programación de ejercicios en prosa. Me ayudó a sentirme más comodo con python. Mi proceso de adaptación fue bastante influenciado por este proyecto y me hizó ver un nuevo enfoque en la computación. Este proyecto tambien me hizo darmw cuenta que disfruto de estar programando y documentando por horas y no tener el mínimo aburrimiento, tal vez un poco de cansancio pero al final se siente que todo valió la pena. Sigo disfrutando de este curso aunque aveces sea díficil.

References

[1] E. R. Jiménez, *Fundamentos de Programación*. 2022.
(1)